**Curso de Git e Github COMPLETO 2023 [Iniciantes] + Desafios + Muita Prática**

**https://www.youtube.com/watch?v=kB5e-gTAl\_s**

**GIT:** usado para fazer versionamento de arquivos, criar, restaurar, atualizar e mesclar as versões deles

**GITHUB:** local para armazenar os arquivos online

# **1. PASSOS PARA PRIMEIRO OU NOVO REPOSITÓRIO:**

1. Instalar o git
2. Abrir o cmd e digitar “git --help ”
3. Criar uma pasta para o projeto
4. Botão direito e selecionar “git bash here”
   1. Abre a linha de comando do git
   2. Digitar comando “git init”
   3. Ver itens ocultos (arquivo “.git” de inicialização do git
5. Digitar **git status** para mostrar quais arquivos estão ou não inclusos no controle de versão
   1. “**On branch ...”**  mostra em qual branch está
   2. “**no commits yet**” não tem nenhuma commit ainda
   3. “**untracked files:**” lista em vermelho os arquivos que não estão incluídos no controle de versão ou se foram modificados

* SE NÃO QUISER ADICIONAR DETERMINADO ARQUIVO VÁ AO **TÓPICO 6**

1. Digitar **git add “nome do arquivo”** vai adicionar esse arquivo ao commit
   1. Digitar **git status**  vai mostrar em verde o arquivo que foi adicionado ao commit (versão atual)
2. Digitar **git add .**  adiciona todos os arquivos que não estão incluídos no commit
3. Depois de adicionar os arquivos criar uma nova versão com **git** **commit -m “mensagem aqui”** na mensagem é uma breve descrição sobre essa versão dos arquivos. Se for a primeira versão pode colocar a mensagem como “commit inicial”
   1. Se for o primeiro commit vai pedir para configurar o git no pc
   2. Comando sugerido para a configuração **git config --global user.email “ email do github”**
   3. Comando para o nome **git config --global user.user “seu nome”**
   4. Depois de configurado fazer o commit novamente **git commit -m “commit inicial”** (no caso do primeiro commit)
   5. Mostra o commit criado e os arquivos que foram adicionados
      1. [branch (root-commit) cod\_versao] mensagem do commit
      2. Qnt de arquivos incluídos, inserção ou exclusão
4. Para enviar o commit para a nuvem (github) digitar o comando **git push**
   1. Se for a primeira versão (commit) vai dar um erro porque precisa definir para onde esse commit será enviado.
   2. Para criar um link para armazenar esse commit.
      1. Ir até o perfil do Github.
      2. Clicar no sinal de **+**  e selecionar para criar um novo **repositório**
      3. Dar um nome para o repositório e uma descrição.
      4. Opção público: deixar o repositório aberto ao público em geral (recrutadores)
      5. Privado: só o dono tem acesso
      6. Adicionar um arquivo **README.file**
      7. Criar o repositório
      8. Copiar o link do repositório criado
   3. Voltar o terminal do GIT e digitar **git remote add origin link-repositorio**.Isso define para onde o commit será enviado
   4. Par enviar de fato digitar **git push**
5. Se for o primeiro envio vai dar um erro porque não foi passado para qual **branch**  do repositório esse commit será enviado
   1. Inicialmente se envia para a branch **master** 
      1. Então copie e execute o comando sugerido **git push --set-upstream origin master**
      2. Se abrir a janela do GitHub entre com a mesma conta que foi passada na configuração **(Passo 8)**

# **2. NOVOS COMMITS**

**Obs.: Em um Git configurado e linkado com um repositório no GitHub**

* + - 1. Comando **git status**  vai mostrar se houve uma alteração no commit e quais arquivos foram alterados (ficam em vermelho).
* SE NÃO QUISER ADICIONAR DETERMINADO ARQUIVO EXECUTE O **ITEM 6**
  + - 1. Comando **git add .** adiciona todos os arquivos alterados no commit
      2. Comando **git status**  vai mostrar em verde os arquivos adicionados para a nova versão
      3. Comando **git commit -m “mensagem”** vai criar um novo commit com uma mensagem de descrição do que foi adicionado nessa versão.

1. [branch cod\_versao] mensagem do commit
2. Qnt de arquivos incluídos, inserção ou exclusão
   * + 1. Comando **git push** vai enviar esse novo commit para o repositório no GitHub.

# **3. HISTÓRICO DAS VERSÕES / COMMITS**

**Commit:** versão/cópia de cada modificação do código, essas versões podem ser restauradas

1. Comando **git reflog** vai listar os commits que foram feitos
2. Para voltar a uma versão anterior, digitar o comando **git reset --hard “id\_versão”** sem as aspas. O id\_versão é o código daquela versão listado no **passo 1.**
   1. Essa restauração deve ser autorizada pelo responsável do projeto

# **4. O QUE É E COMO CRIAR UMA BRANCH**

**Branch:** São caminhos ou versões diferentes do código (pastas gerais com cópias de teste e versão final)

## **Tipos de branch:**

* **Master:** é a branch/caminho principal, onde vai estar a versão atual e funcional do programa como um todo, **ex:** versão 1.0, versão 1.1, versão 1.2, etc. Em cada versão tem uma correção ou funcionalidade nova para o programa.
* **Branch Paralelas ou Temporárias:** são branch de produção ou correção de código. Depois de finalizado o código dessa branch vai ser **analisado pelo responsável do projeto/programa** e unificado (processo de merge) com a branch principal (master).

## **Criação de Branch:**

1. Comando **git branch** retorna as branch do projeto.
   1. O asterisco \* na frente do nome da branch indica que essa branch está aberta
2. Comando **git branch “nome da branch”**  (sem aspas) vai criar uma nova branch com o nome informado.
   1. Geralmente para a segunda branch é usado o nome ‘staging’ que indica que é uma branch de teste. Comando **git branch staging**. Mas pode ser o nome da funcionalidade ou correção.
3. Para mudar de branch execute o **passo 1** e depois o comando **git checkout nome\_branch** 
   1. Mensagem de confirmação da mudança “Switched to branch ‘nome\_branch’ ”
   2. O comando **git checkout -b nome\_branch\_nova nome\_branch\_master**  vai criar e acessar uma branch nova a partir da branch principal (master)
4. Ao enviar pela 1 vez um commit para o repositório na branch nova deve ser usado o comando **git push --set-upstream origin nome\_branch**. Assim configura o endereço de envio desse commit na branch nova.

# **5. PROCESSO DE MERGE**

O processo de merge vai unir a branch principal com uma branch ‘paralela’, assim incrementando o código produzido em teste no código do programa final.

1. Para atualizar a branch principal, primeiro deve acessar ela com o **passo 4.3**.
2. Com o **passo 4.1** confirme que está na branch principal.
3. Depois pegue a versão atual da branch principal do repositório com o comando **git pull**
   1. Isso irá atualizar a branch principal na máquina e garantir que o merge será feito com a última versão da branch principal
4. Depois de confirmar digite o comando **git merge nome\_branch\_paralela**, com isso os códigos e arquivos da branch paralela serão incorporados na branch principal.
   1. Vai ser listado os arquivos que foram alterados, e a qnt de inserções e exclusões feitas
5. Comando **git push**  para enviar esse merge para branch principal do repositório no GitHub.

# **6. GIT IGNORE**

Ignora alguns arquivos não os enviando para o repositório ao fazer o **push**

1. Antes de fazer um novo commit com o **passo 2.2**
2. Excute no terminal do Git o comando **touch .gitignore**
   1. Vai ser criado um arquivo com a extensão **.gitignore**
   2. Ao abri-lo com o bloco de notas digite quais pastas e arquivos não serão inclusos no commit.
   3. Para pastas digite o nome da pasta seguido por uma barra, ex: **pasta/**
   4. Para arquivo digite o nome e a extensão do arquivo, ex **arq.txt**
   5. Salve o arquivo e execute o comando **git status** e depois o comando **git add .**  para adicionar esse arquivo **.gitignore** ao commit
   6. Faça o **commit** e **push** normalmente.

**PULL REQUEST:** Solicitação de aprovação ao commit enviado.

**minuto 41:09**

[**https://github.com/Dev-JoaoM/Cursos**](https://github.com/Dev-JoaoM/Cursos)

**### 💻 Resumos das Aulas**

**|Aulas|Resumos|**

**|-----|-------|**

**|Autenticação via token| [Aula](https://web.dio.me/course/versionamento-de-codigo-com-git-e-github/learning/3d13d85f-2508-4396-9657-4643d3302c79?back=/track/coding-future-vivo-python-ai-backend-developer&tab=undefined&moduleId=undefined)|**

**|Autenticação via Chave SSH| [Aula](https://web.dio.me/course/versionamento-de-codigo-com-git-e-github/learning/a53b7d6e-d7a2-40de-a8f9-cc30b42fc93d?back=/track/coding-future-vivo-python-ai-backend-developer&tab=undefined&moduleId=undefined)|**